Automatically Generating IsiZulu Words From Indo-Arabic Numerals

Zola Mahlaza and **Tadiwa Magwenzi** and **C. Maria Keet**
University of Cape Town
South Africa
{zmahlaza,mkeet}@cs.uct.ac.za, MGWTAD001@myuct.ac.za

Langa Khumalo
SADiLaR, Northwest University,
Potchefstrom, South Africa
langa.khumalo@nwu.ac.za

Abstract

Artificial conversational agents are deployed to assist humans in a variety of tasks. Some of these tasks require the capability to communicate numbers as part of their internal and abstract representations of meaning, such as for banking and scheduling appointments. They currently cannot do so for isiZulu, due to the lack of speech and text data and the complexity of the generation due to dependence on noun that is counted. We solved this by extracting and iteratively improving on the rules for speaking and writing numerals as words and creating two algorithms for it. Evaluation of the output by two isiZulu grammarians showed that six out of seven number categories were 90-100% correct. The software was used with an additional set of rules to create a large monolingual text corpus, made up of 771,643 sentences, to enable future data-driven approaches.

1 Introduction

Artificial conversational agents are frequently deployed to interact with humans and execute simple tasks on their behalf. For such agents to be useful for people who speak South African languages, various Natural Language Processing (NLP) tools need to be built. For instance, if an isiZulu speaker is negotiating with a digital assistant to book a restaurant table, it may present a feasible option as follows:

Indawo yokudlela iX inetafula labantu aba-2 ngomhla ka-25 (IsiZulu)

'Restaurant X has a table for 2 people available on the 25th'

where the underlined parts are used to mark agreement between numbers and their subjects in the sentence: the *aba-* is determined by the noun class of *abantu* 'people', the subject of the number 2, and the ka is determined by the range of the number that follows it. Since isiZulu, the largest South African language by L1 speakers, has an agglutinating morphology and agreement markers in numbers and other parts of speech, the inclusion of Indo-Arabic numerals in text often yields hard-to-read text, especially if the underlined prefixes are omitted, since then the text is grammatically incorrect. Then, the reader has to figure out what is being counted, as it is not encoded in the text as it should be. This issue can also lead to inconsistencies in orthography (Ndimande-Hlongwa, 2010, p218) and confusion due to differences in how the reader ought to interpret the text in the absence of an explicit concord. It can be addressed by presenting numbers as words instead of numerals, which also will solve this gap in text-to-speech systems. However, that is currently impossible to do, because there are no comprehensive algorithms to convert numerals into their equivalent word form. There are also no large datasets that can be used to build seq2seq text normalisation models for the task.

It is, however, not only a case of agreeing prefixes. Consider the verbalisation of the number 2, *-bili*: it renders as *abantu ababili* for 'two people' and *izinja ezimbili* for 'two dogs', among many forms. *Ababili* is formed by appending the subject concord *aba-* to the stem *-bili*. *Ezimbili*, however, was subjected to phonological conditioning rules when combining the subject concord *ezin-* with *-bili* hence the word has an *-m-*. The form depends on the noun class of the noun it quantifies over, which is indicated with the underlined prefixes. This, in turn, is due to the noun class system emblematic of the Niger-Congo B (NCB) languages[1] (Herbert and Bailey, 2002); *abantu* is in noun class 2 whereas *izinja* is in noun class 10. IsiZulu has 17 noun classes. The formation of such words requires understanding of the numerical categories, the patterns for each category, and the resolution of the appropriate prefix for the various categories. Afterward, rules for combining a variety of morphemes need to be applied to obtain the final word.

[1]Some historical sources use the term 'Bantu' languages.

In this paper, we propose the first solution to this problem of generating words from Indo-Arabic numerals for 'standard' isiZulu. We collected, analysed, tested, and formalised the text generation rules and designed and implemented two new algorithms that convert numerals to words. The algorithms cover cardinal, ordinal, and set-of-items numerals, and numerical adverbs, which generate noun phrases such as, e.g., *ama-apula ayisishiyagalolunye* 'nine apples' (with *-shiyagalolunye* 'nine'), *ama-apula wesishiyagalolunye* 'ninth apple', *ama-apula omasishiyagalolunye* 'all nine apples', and *ngithenge ama-apula kasishiyagalolunye* 'I bought apples nine times', respectively.

To demonstrate utility of the algorithms, we developed a sentence generation system for isiZulu, focusing on handling various numerical types and generated a corpus of 771,643 grammatically correct sentences. This is the first publicly accessible isiZulu dataset of its size that is not based on the Bible, government documents, or technical manuals. It contains ten times more sentences than the clean NCHLT monolingual isiZulu dataset (Eiselen and Puttkammer, 2014) that is widely used.

These algorithms were developed and implemented using two iterations; for each iteration, we used grammar literature to identify the linguistic categories of numbers, determined the patterns for forming words in each category, and used our linguistic knowledge. Our final evaluation is expert-focused, relying on two isiZulu grammarians, working collaboratively, to ascertain the accuracy of the algorithms' output. It showed that five of the seven number categories had 100% valid output, one 90%, and one had 30% correctness due to a change in concord.

The remainder of the paper is structured as follows: Section 2 introduces key linguistic properties of isiZulu to demonstrate why generating text from numerals is not trivial and it also discusses existing Natural Language Generation (NLG) work with a special emphasis on isiZulu. Section 3 presents our novel algorithms and the procedure followed for their development. Section 4 presents the expert-driven evaluation and results, Section 5 discusses the results and demonstrates the utility of the algorithms via generating a large corpus that can be used in creating data-driven models, and Section 6 concludes.

2 Natural language generation and isiZulu

NLG research focuses on generating natural language text from a variety of different inputs (e.g., (van der Lee et al., 2018; Gkatzia et al., 2016)). With respect to NCB languages, a few NLG systems and algorithms have been developed, notably grammar rules to generate texts in a specific subject domain (Byamugisha et al., 2016a; Mahlaza, 2018) or for a specific task, such as verbalisers for maths equations, ontologies, or language learning exercises (Keet et al., 2017; Byamugisha, 2019; Smith, 2020; Mahlaza and Keet, 2020; Gilbert and Keet, 2018). To the best of our knowledge, there are no existing algorithms, let alone implementations, that can be used to programmatically convert numerals to isiZulu words. There only exists a grammar fragment to verbalise numbers in the range 1-99 in the *WeatherFact* grammar (Marais, 2021a).

Relevant text-to-speech work include Marais et al.'s (2020) grammar that has the type *SmallNumber* to verbalise numbers in isiZulu. There is insufficient documentation of the grammar, but the dataset used to create it shows that its capability is likely limited to numbers between 1-10 (Marais, 2021b). Schlünz et al.'s (2017) work has greater coverage for isiZulu, but they only generate ordinal numbers that agree with nouns from noun class 3, the coverage is limited to numbers up to 100 based on our analysis of the documentation, and there is insufficient detail of the number generation process other than regular expressions with modulo arithmetic.

This lack of capability is partly due to the complexity of the language, and of the number system specifically. IsiZulu is a NCB language, most of which possess a highly agglutinating morphology, i.e., words are formed through combining multiple morphemes. All nouns belong to a noun class, which is used to make a part-of-speech in agreement with a noun. The number of noun classes in a NCB language varies depending on the language and the chosen noun classification system. For instance, Grout's (1893) classification system has eight noun classes, whereas the most used classification system, originally due to Meinhof (Katamba, 2014), has 17 noun classes for isiZulu.

To obtain agreement in isiZulu sentences, the class of the noun that is qualified by the number is first identified, its concord(s) (i.e., special morphemes for marking agreement) are identified, and then used together with other morphemes to form the final string for the qualifying number. This process may require phonological conditioning rules to ensure that one obtains a valid word; e.g., aforementioned *ezimbili* 'two', because isiZulu disallows the voiced alveolar nasal *n* to be followed by the voiced bilabial implosive *b* and so noun class 8's *n* of the *ezin* concord is changed to *m*.

Thus, there is still a need for a comprehensive algorithm that can verbalise numbers, both when they agree with a noun or on their own. Especially since there are no existing parallel datasets that can be used to train a number-to-text model[2].

3 Verbalising numbers

The algorithms were created by codifying rules from grammar literature over two iterations. All the linguistic knowledge was extracted from (Wilkes and Nkosi, 2012; Stuart, 1940; Grout, 1893) and supplemented with the first author's knowledge as a researcher who works with isiZulu.

Due to space limitations, we discuss key aspects in the remainder of this section; the complete set of rules are available as supplementary material in the Appendix.

Number categories We chose to support only numbers within the range 0-9,999 for the numeral categories shown in Figure 1, as the use case motivation was in the context of building a personal finance digital assistant that supports isiZulu and the range was sensible for the target audience.

Patterns and rules for using them The high-level patterns that were extracted from the literature are listed in Appendix A; e.g., Pattern 1c for cardinal numbers:

adj.conc-(*yi*|*ngama*)-*shumi* ((***ama***|***ayisi***)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?

where **adj.conc** is the adjectival concord, *shumi* 'ten', and **stem** the stem of the number that is grammatically a noun in isiZulu. The patterns still require further assessment to determine *when* to use which pattern in each category, *where* to use which morpheme for a segment when there are multiple options, and *when* an optional segment should be included. For instance, for Pattern 1c, there is no information yet when to use **-yi-** or **-ngama-** in the first word. Similarly, when verbalising the number 5 as a cardinal number, the Patterns 1a-1h do not include with of those 8 rules is the one to apply in a particular case.

The pattern selection for each category is based on the range of the number and whether it has to include an agreement marker. The ranges supported by each pattern are included in Appendix B; that is, which pattern apply to numbers $0 < n < 10$, $10 \leq n < 100$, and so on. Some patterns include an adjectival or possessive concord; those that have concords are only used when verbalising numbers that need to agree with a noun. For instance, the cardinal number 2 is verbalised as *ababili* 'two' when it agrees with nouns in noun class 2 and it is *isibili* without agreement marker.

Once a pattern is selected for the range and agreement marker, there is another set of rules to select an appropriate morpheme for the pattern parts that have multiple values (the parts that are coloured in the patterns), and then rules for deciding whether to include the optional segments.

The pattern selection is decided using the rules described in Appendix B. We describe one of those rules here, for brevity. The stems that are used for numbers that are less than 10 (i.e., **stem**$_{number<10}$) may be preceded by an optional segment (e.g., see Patterns 1b and 3a) and these segments are only included if the first number to be verbalised is in the range of [6,9] inclusive. For instance, if we take Pattern 1b to verbalise the number 5 for a noun in noun class 2, it generates *abahlanu* (the *-yisi-* is omitted), whereas the number 6 (still with noun class 2) is verbalised as *abayisithupha*—with-*yisi*-, instead of *abathupha*—since it belongs to the [6,9] range.

Pattern use illustration We demonstrate how the patterns can be used to verbalise the cardinal numbers 25 and 26 when they agree with nouns from class 2. The patterns must output *abangamashumi amabili nanhlanu* 'two tens and five' (i.e., twenty-five) and *abangamashumi amabili nesithupha* 'two tens and six' (i.e., twenty-six). In all the generated texts, the first word is a reference to tens, the second word references the number of tens (i.e., two), and the third word references the remainder that is left after subtracting the two tens (i.e., 5 and 6, respectively). The final morphemes that are chosen for each word are given in Table 1, which are explained in the remainder of this paragraph.

We use the pattern selection rules in Appendix B to identify the rule:

[2]A list of relevant isiZulu datasets can be found at `https://github.com/masakhane-io/masakhane-community/blob/master/list-of-datasets.md`.

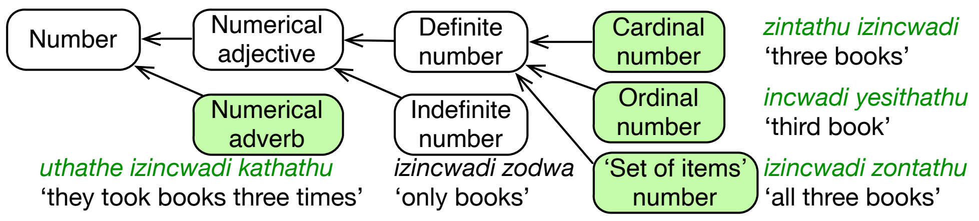

Figure 1: A taxonomy of the several types of numbers in isiZulu (Adapted from (Grout, 1893)). Green shaded boxes indicate the categories covered by our algorithms.

Table 1: Pattern used to verbalise the numbers 25 and 26 when they agree with noun class 2, and, for comparison, the components and output for 14 and 17 when in agreement with noun class 4, and 84 and 87 with noun class 8. For each number, the values for each slot have been inserted and the appropriate segment is chosen when there are multiple options.

	First word 'agreement tens'			Second word 'amount of tens'		Third word 'and remainder'	
Pattern	**adj.conc**	*(yi|ngama)*	*shumi*	((*ama*|*ayisi*)	**stem**$_{count10}$)?	(*na*	**stem**$_{number<10}$ | **noun**)?
	Agreement with noun class 2						
25	aba	ngama	shumi	ama	bili	na	hlanu
	abangamashumi amabili nanhlanu						
26	aba	ngama	shumi	ama	bili	na	isithupha
	abangamashumi amabili nesithupha						
	Agreement with noun class 4						
14	emi	yi	shumi	∅	∅	na	ne
	emiyishumi nane						
17	emi	yi	shumi	∅	∅	na	isikhombisa
	emiyishumi nesikhombisa						
	Agreement with noun class 8						
84	ezi	yi	shumi	ayisi	isishiyagalombili	na	ne
	eziyishumi ayisishiyagalombili nane						
87	ezi	yi	shumi	ayisi	isishiyagalombili	na	isikhombisa
	eziyishumi ayisishiyagalombili nesikhombisa						

1. First, both numbers are in the range [10,100], second, they have agreement markers, third, they are cardinal numbers, hence Pattern 1c is applicable.
2. The first word in the pattern **adj.conc**-(*yi*|*ngama*)-*shumi*) and the following optional segments, i.e., ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?) have morphemes whose value must be chosen from two possible values (in pink and blue colour).
3. To form the first word (from left-to-right), we start by resolving the adjectival concord,[3] which is *aba*- for noun class 2. We then use Table 2 to determine the prefix for the second morpheme: **Segment 2**, 10/100 column, plural, agreement, cardinal, which gives us -*ngama*-. The first word's third morpheme is -*shumi* for every input. So, the first word becomes *abangamashumi*.
4. For the second word on multiples of ten, we start with Table 3 to resolve the value of the morpheme: for the 10/100 row, and with 2 being in the [2-5] range, the prefix is *ama*-. For the second morpheme, the stem is -*bili* 'two' since there are two tens in the input, resulting in *amabili*.
5. For the last word, we start with the conjunction *na*- 'and' irrespective of the remainder and then either i) use the stem of the number that is associated with the remainder after removing the two 10s, for numbers in the range [1-5], or ii) use the stem to form a noun for the remainder, for numbers in the range [6-9]. So, with a remainder of 5, we use the stem -*hlanu* 'five' to obtain *nanhlanu*, and for 6, being *isithupha*, we obtain *nesithupha* after phonological conditioning, applying the *na* + *i*- → *ne*- rule.

As mentioned before, combining morphemes may activate phonological conditioning rules, which is a separate issue not considered here (see further below).

[3] https://github.com/mkeet/MoRENL/blob/main/resources/ZuluConcordsListof22.pdf

Table 2: List of possible prefix values used for the segments that are used to construct the strings that refer to special multiples of ten. We use the ∅ symbol to denote that a prefix is not applicable for a category. Abbreviations used: Plural = Pl., Singular = Sg.

	10/100		**1000**		**10/100**		**1000**		**Category**
	Sg.	**Pl.**	**Sg.**	**Pl.**	**Sg.**	**Pl.**	**Sg.**	**Pl.**	
Agreement	∅	∅	∅	∅	*yi*	*ngama*	*yi*	*yizi*	Cardinal
	∅	∅	∅	∅	*i*	*ama*	*i*	*izi*	Ordinal
	∅	∅	∅	∅	*li*	*ma*	*i*	*yizi*	Set-of-items
No Agreement	∅	∅	∅	∅	*i*	*ama*	*i*	*izi*	Cardinal, ordinal, set-of-items
	kali	*kanga*	∅	∅	*i*	*ama*	*i*	*izi*	Adverb
	Segment 1				**Segment 2**				

Table 3: List of prefixes used in the word that count the number of multiples of 10s (e.g., the second word in *amakhulu amathathu* 'three hundred'). The value of 1 is not included in the ranges (second column), because the segments with the prefixes are not included when there is only one 10, 100, or 1000.

Quantified number(s)	**Value/range of count**	**Prefix**
10/100	6-9	**ayisi-**
10/100	2-5	**ama-**
1000	2, 4	**ezim-**
1000	3 or 5	**ezin-**
1000	6-9	**eziyi-**

Algorithms Using the patterns and rules described in the previous sections as a basis, we created Algorithms 1 and 2 (see supplementary material) to capture all the necessary information. Algorithm 1 is used to verbalise numbers that do not have an agreement marker while Algorithm 2 is created to generate numbers have one. In both algorithms, we use a plus sign to denote the concatenation of morphemes, and the symbol **mod** to denote the modulo arithmetic operator. This operation is not a simple appending of morphemes since it may activate the necessary phonological conditioning rules. We used the phonological conditioning rules described in (Mahlaza and Keet, 2020) and extended them with rules for combining nasals and fricatives (Raper, 2012; Naidoo, 2005). All these auxiliary rules are implemented in a Java-based grammar engine for Nguni languages[4]. The algorithms for the text generation for numerals were implemented using Java, they rely on the previously grammar engine for phonological conditioning, and the implementations are available as supplementary material[5].

To demonstrate it, we use the generation of text for the ordinal 105 using Algorithm 2 with nouns from class 8 to produce *zekhulu nanhlanu*. When tracing the algorithm, and 'line(s)' here referring to the lines in Algorithm 2:

1. The closest multiple of 10 is 100 (lines 15-16) with a remainder of 5 (line 19).
2. Since the value is ordinal (line 23), the chosen concord is *za-* (line 25), the prefix and stem are *-i-* and *-khulu* respectively (line 29), and
3. they are combined to form *zekhulu* for the first word where the rule a+i → e is applied to eliminate the prefix and the *-a-* from the concord (rule is encoded in the grammar engine).
4. After removing 100 from the input, the remainder is 5 (lines 33-41) and
5. it is less than six, therefore its stem *-hlanu* is combined with the conjunction *na-* (line 34) to form *na<u>n</u>hlanu* where the *-n-* is introduced by phonological conditioning.

Related to the previous example, when using Algorithm 1 to generate text for the number 84 when there is no agreement marker, the output is *kangamashumi ayisishiyagalombili nane* 'eighty-four times'. Specifically, and with 'line(s)' referring to the lines in Algorithm 1):

1. The algorithm first establishes that the category is a numerical adverb and that the closest multiple of 10 is 10 (lines 17-20) with a remainder of 4 (line 21).
2. Since there are 8 tens (line 22), hence, the multiples are plural (line 23), the algorithm then retrieves the prefixes *kanga-* (i.e., **Segment 1**), *-ama-* (line 25) and stem *shumi* to produce *kangamashumi* (line 25) where a phonological conditioning rule removes the duplication

[4] `https://github.com/AdeebNqo/NguniTextGeneration`

[5] `https://github.com/KEEN-Research/ZuluNum2Text`

of *-a-* when combining of *kanga* + *ama*.

3. Following that, since there are multiple tens (line 29-30), the algorithm combines *ayi-* with *-isishiyangalombili* to form *ayisishiyangalombili* (line 30) for the second word.
4. The remainder is 4, hence, the numerical stem *-ne* is selected and combined with the conjunction *na-* to form *nane* (line 34), forming the third word.
5. The three words are then combined to form the final output.

4 Evaluation of the algorithms

The aim of the experiment is to evaluate the accuracy of the algorithms we developed. The entire process of algorithm developed up to good quality output took two iterations, illustrated in Figure 2, but due to space limitations, we only report on the evaluation of the second, and final, iteration.[6] It is also with human judgements, since there is no corpus to check the numbers against. To maximise the likelihood of being able to determine why generated texts are grammatically incorrect, if the need were to arise, we chose to rely on two isiZulu grammarians to collaboratively evaluate the texts instead of only using isiZulu speakers. We describe the methods and results in this section and discuss them in Section 5.

4.1 Materials and Methods

We sought to create a survey that is made up of numbers that are representative of the various number categories and not biased in favour of a specific noun class. This was balanced against keeping the number of generated texts as low as possible to avoid obtaining untrustworthy judgements due to fatigue. As such, we randomly sampled one noun and then used it to generate numbers that have agreement markers across the relevant number categories. We could not reasonably include all the numbers and for every noun classes since that would have meant that the grammarians would have to evaluate 519,948 texts (i.e., (9,999 numbers * 16 noun classes * 3 categories of numbers with agreement markers) + (9,999 numbers * 4 categories of numbers without agreement markers)). The validity of the generated strings is not compared, at least not directly, with strings from another system or algorithm since no comparable system or algorithm exists. We will return to this point in Section 5.

We generated 70 texts by first sampling five numbers from the list of numbers that have unique word stems (see Section 3) and another five from numbers that do not have unique stems, in the range between 10 and 10,000. We then verbalised those ten numbers for the cardinal, ordinal, set-of-items, and adverb categories such that they are not in agreement with any noun; the resulting number of strings are listed in Table 4 in the first three columns.

Table 4: List of the 70 texts judged by isiZulu grammarians, separated by number category and agreement, and the percentage of valid texts. Abbreviation(s) used: Agreement = Agr., Percentage = Pct., Number = Num.

	Category	Noun Class	Num. texts	Pct. valid
No Agr.	Cardinal	n/a	10	100%
	Ordinal	n/a	10	100%
	Set-of-items	n/a	10	100%
	Numerical adverb	n/a	10	100%
Agr.	Cardinal	2	10	100%
	Ordinal	2	10	90%
	Set-of-items	2	10	30%

In order to generate numbers that agree with some noun, we selected the first plural noun in the first section of an English-IsiZulu dictionary (de Schryver, 2015). We used the sampled noun *ababhali* 'writers' from noun class 2 to verbalise the selected numbers for all the categories that have agreement markers.

The 70 texts were packaged into a single spreadsheet and collaboratively analysed by the two grammarians to determine whether each of them was valid or invalid, or state whether they were uncertain. If the verbalisation was invalid, they were asked to provide optional comments to describe the source of the error. Since the evaluation was collaborative, inter-annotator agreement scores were not applicable. They were recruited through direct invitation by email, from our pool of prior collaborators and evaluators.

4.2 Results

The aggregated results of the judgments made by the grammarians are summarised in Table 4 in the last column. They are overwhelmingly correct, except for the set-of-items category.

[6]Details of the first iteration can be found in the report by Moraba (2021).

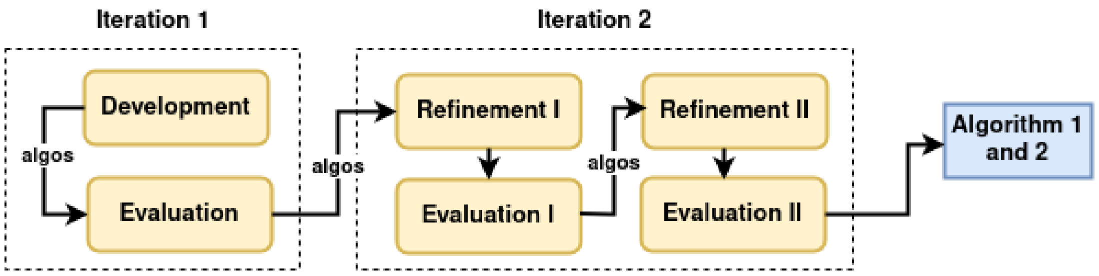

Figure 2: Steps taken to develop the algorithms. Evaluation in iteration 1 relied on L1 and L2 isiZulu speakers for evaluation while iteration 2 relied on grammarians.

Table 5: List of old and updated prefix values used for the segments that are used to construct the strings that refer to special multiples of ten. Prefixes are grouped for patterns that agreement markers and patterns with no markers. The ∅ symbol denotes an inapplicable prefix. Abbreviations used: Plural = P., Singular = S.

	10/100		**1000**		**10/100**		**1000**	
-	**S.**	**P.**	**S.**	**P.**	**S.**	**P.**	**S.**	**P.**
Old	∅	∅	∅	∅	*li*	*ma*	*i*	*yizi*
New	∅	∅	∅	∅	*yi*	*ngama*	*yi*	*yizi*

Error analysis shows that the set-of-items category received a low percentage because of an incorrect use of an adjectival concord instead of possessive concord in patterns that "appl[y] more to numbers above five than those below" (Grout, 1893). The grammarians also pointed out that the *Segment 2* values used in the set-of-items patterns are also incorrect. We have updated it accordingly; the changes are listed in Table 7. After making these changes, we used the grammarians' comments, where they specified the correct forms, to confirm that the changes resolve all the errors.

The second, and minor, issue concerned ordinals with agreement marker, obtaining 90% correct. The error analysis shows that only one number was deemed invalid, which was due to the use of -*isi*- instead of -*i*- when forming a noun using the stem -*khulu*, resulting in *besikhulu* instead of the expected *bekhulu*. This was caused by missing a rule that is not explicitly mentioned by Grout (1893, pg90). Grout (1893) specifies that nouns are formed by prefixing *isi*- to the stem and we were able to determine, using Grout's examples of nouns (Grout, 1893), that the number 10 is an exception as it uses *i*-. This *i*- exception turned out to apply also to 100 and 1000.

Therefore, we updated Table 2's column that specifies the prefix values when there is agreement in a word. The old and new values for the set-of-items category are given in Table 5. This now allows the generation of two 100s (i.e., 200), as set-of-items, when it agrees with noun class 8 as *ezingamakhulu amabili* instead of *ezimakhulu amabili* (updated and old prefix values, respectively, underlined). This also induced a minor change to lines 17-42 of Algorithm 2 so that it now uses the possessive concord and basic prefix. The validity of the change was confirmed by comparing the output to the correct value provided via comments by the grammarians.

5 Discussion

The 'old-fashioned' laborious approach of consulting documentation and encoding it has been shown to work well for the isiZulu numbers, considering the results of the final evaluation. The overall process was hampered by a lack of recent and relevant books describing the language's grammar, which required combining material from comprehensive dated books, recent language learning books, our isiZulu expertise, and iterations with intermediate testing. The multiple iterations in algorithm development were mainly due to incorporating changes throughout time regarding orthography and noun classification and subsequent refactoring of components, specifically regarding phonological conditioning rules.

Even though the grammar books used are dated, they were still valuable sources of linguistic information to understand the main mechanism of generating words from numbers. Specific issues that surfaced during development were:

- Old textbooks use -*t*- instead of -*th*- hence they use *katatu* instead of *kathathu* 'four times'. However, only -*th*- is used in modern isiZulu hence an output of *katatu* will be deemed incorrect.
- Grout (1893) uses adjectival concords for

marking agreement in set-of-items numbers that are greater than 5. However, these are judged to be invalid by two grammarians. This is likely because Grout's grammatical construction is outdated.

5.1 Comparison to related work

We now turn to compare our algorithms to existing work: Marais's (2021a) recent grammar of isiZulu focuses on a proof-of-concept question answering system and possesses a small module for numbers, covering a subset of those that can have agreement markers for three nouns (i.e., *imizuzu* 'minutes', *amahora* 'hours', and *izinsuku* 'days'). The numbers are only generated to refer to a small number of minutes, hours, or days in the context of a Q&A about the weather. Also, while Schlünz et al.'s (2017)'s coverage is broader than Marais' work, it is also limited to 100—far less than ours. We thus have surpassed the state-of-the-art, since we have created the first well documented and high coverage algorithms.

For comparisons to other existing work, we considered relying on existing neural machine translation (MT) systems that support isiZulu (e.g., (Nyoni and Bassett, 2021; Sefara et al., 2021; Chiguvare and Cleghorn, 2021)). However, that is infeasible because the models are not controllable (i.e., one cannot specify that they want to generate numbers that belong to a specific category as opposed to another); hence, they cannot generate text for all the appropriate number categories listed in Figure 1. Moreover, developing a controllable model from scratch is impractical at present because there is no large parallel corpus for Indo-Arabic numeral verbalisation in isiZulu. Re-purposing MT models for numeral verbalisation, a task for which they were not created, does not yield a sensible baseline. We considered comparing our algorithms to automatically translated English-to-isiZulu verbalisations. In such systems, one would have to generate English verbalisations for each category and then translate the output to isiZulu using an MT model. We operationalised this by creating an ensemble model that first verbalises numerals to English via templates and then translates them into isiZulu via SMaLL-100 (Mohammadshahi et al., 2022), however, none of the model's output was judged, by the first author, to be valid. The model 'hallucinated' nouns that are unrelated to the input numeral (*ikhaya* 'home' from cardinal 2), there was invalid repetition of verbalised number ('*elishumi elishumi* ... 'ten ten ten ...' from cardinal 6,718), etc. The approach is not sensible because it introduces complications for which it is not easy to control and outside the scope of our research. For instance, the following choices make a difference to the quality of the output: choice of language to use as the source, the length of the input text, what nouns are present in the English input, etc.

Since large language models (LLMs) have demonstrated remarkable performance in a variety of tasks, we considered comparing our algorithms to LLMs, however, we deemed such a comparison to be out of scope since additional work is required to establish which model(s) qualify as suitable baselines and what configuration to use when generating text. This is because it has been demonstrated as part of IrokoBench (Adelani et al., 2024), a benchmark on natural language inference, mathematical reasoning, and knowledge-based QA for 16 African languages, that while closed LLMs (e.g., GPT-4o) tend to outperform most open LLMs, this is not consistent across all tasks. In addition, while there are cases where performance gains are seen when prompts are authored in the language to be generated instead of English, this is also not consistent across tasks. As such, for the task under consideration, additional work is still required to establish the best model(s) and their optimal settings/setup prior to comparing them to the proposed algorithms.

5.2 Corpus creation exploiting the rules

Therefore, to demonstrate the utility of the algorithms for data creation, we gathered the pluraliser and its set of 218 nouns with noun classes and their plurals (Byamugisha et al., 2016b), verb conjugation rules from (Keet and Khumalo, 2017b,a), and the idea of the exercise generator of (Gilbert and Keet, 2018) to generate a corpus for numbers that may be of use to augment data-driven approaches. Specifically, there is the basic noun phrase generation for all of the numbers 0-9,999 without agreement, and then with agreement for each noun class. They can be paired with nouns, such as 'three books', 'three apples' etc. to assist machine/deep learning models to learn the agreement co-occurrences. Third, phrases are constructed by stringing guaranteed to be semantically acceptable combinations for three bags of words using templates, partially thanks to the semantics of the noun classes (e.g., noun class 1 contains only humans and the roles they play). Three examples

$NP_{\text{select noun from nc1}}$ $V_{\text{select verb from: buys/reads/shelves/reviews/sells}}$ $NP_{\text{object = books}}$ <generate cardinal number between 0 and 10000>
$NP_{\text{select noun from nc1}}$ V_{buys} $NP_{\text{object from nc6, 8, or 10}}$ <generate cardinal number from 0 to 10000>
$NP_{\text{select noun from nc1}}$ $V_{\text{select verb from: buys/reads/shelves/reviews/sells}}$ $NP_{\text{object = books}}$ <generate numerical adverb from 0 to 10000>

Figure 3: Examples of the parameterised templates. Noun class 1 consists of nous that have humans as referents, and for noun class 6, 8, and 10, it takes a subset concerning the objects and utensils.

Table 6: Number of sentences that include each category of generated numbers in the corpus created from the rules and bags of words.

Category	**Number of sentences**
Cardinal	171,986
Set-of-items	149,133
Ordinal	193,088
Numerical adverbs	257,436
Total: 771,643	

of such patterns are illustrated in Fig. 3. Likewise, one can create other variants and generate a Cartesian product for subjects, verbs, and a number of objects.

We implemented this in a re-deployable tool, the IsiZulu Sentence Generator, which is a Java-based tool designed to generate sentences in the isiZulu language by combining verbs, nouns, and numbers, calling `ZuluNum2TextCMD.jar` from the generic implementation (see Footnote 5). The tool reads data from a CSV file containing verb roots, nouns, and noun classes, processes the data, and generates sentences based on those predefined templates. The generated sentences are then written to CSV files for further use. We generated a corpus with 7,533,595 tokens and the number of sentences generated with the small vocabulary, for each category of numbers, are given in Table 6. The code and corpus are available as supplementary material.[7]

The complete sentences with the written-out numbers may then also be used to train text-to-speech algorithms that then can be deployed in the prospective banking-cum-financial literacy app from the motivational use case and other ones, such as the AwezaMed medical app (Marais et al., 2020). One trivially can add more nouns, their noun classes, and verbs in the lexicon sets used for generation to create a larger corpus, or to generate the corresponding sentences in another language to generate a parallel corpus for training, if needed.

[7] https://github.com/KEEN-Research/IsiZuluSentenceGenerator/

6 Conclusion

Based on collected rules for speaking and writing numerals, algorithms for automating this transformation were designed and evaluated. The categories of numerals covered by the algorithms include ordinals, cardinals, collections, and numerical adverbs and they include markers for agreement with noun classes where applicable. The evaluation of the final algorithms, after extending coverage and phonological conditioning, by two isiZulu grammarians showed that 6 of the 7 categories of numerals have 90%-100% valid output. By combining extant open sourced rules with the ones developed in this work, we created a corpus of 771,643 sentences with a total of 7,533,595 tokens (1,086 unique) to facilitate data-driven NLP approaches.

Future work includes extending the range of the covered numbers beyond 0-9,999 and using the algorithms to build a tool that can generate isiZulu text from mathematics equations and determine their impact on learning with a larger number of people to assess the algorithms' quality and utility. In addition, we will also investigate the use of neural models as adaptable methods for verbalisation.

Acknowledgements

This work was financially supported in part by the National Research Foundation (NRF) of South Africa (Grant Numbers 120852 and CPRR23040389063). We also acknowledge Junior Moraba for his contributions to Iteration 1 of the development.

References

David Ifeoluwa Adelani, Jessica Ojo, Israel Abebe Azime, Jian Yun Zhuang, Jesujoba O. Alabi, Xuanli He, Millicent Ochieng, Sara Hooker, Andiswa Bukula, En-Shiun Annie Lee, Chiamaka Chukwuneke, Happy Buzaaba, Blessing Sibanda, Godson Kalipe, Jonathan

Mukiibi, Salomon Kabongo, Foutse Yuehgoh, Mmasibidi Setaka, Lolwethu Ndolela, Nkiruka Odu, Rooweither Mabuya, Shamsuddeen Hassan Muhammad, Salomey Osei, Sokhar Samb, Tadesse Kebede Guge, and Pontus Stenetorp. 2024. Irokobench: A new benchmark for african languages in the age of large language models. *Preprint*, arXiv:2406.03368.

Joan Byamugisha. 2019. *Ontology verbalization in agglutinating Bantu languages: a study of Runyankore and its generalizability*. Ph.D. thesis, Department of Computer Science, University of Cape Town, South Africa.

Joan Byamugisha, C. Maria Keet, and Brian DeRenzi. 2016a. Tense and aspect in Runyankore using a context-free grammar. In *Proceedings of the Ninth International Natural Language Generation Conference, September 5-8, 2016, Edinburgh, UK*, pages 84–88. Association for Computational Linguistics.

Joan Byamugisha, C. Maria Keet, and Langa Khumalo. 2016b. Pluralising nouns in isizulu and related languages. In *Computational Linguistics and Intelligent Text Processing - 17th International Conference, CICLing 2016, Konya, Turkey, April 3-9, 2016, Revised Selected Papers, Part I*, volume 9623 of *Lecture Notes in Computer Science*, pages 271–283. Springer.

Paddington Chiguvare and Christopher W Cleghorn. 2021. Improving transformer model translation for low resource South African languages using BERT. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8.

Gilles-Maurice de Schryver. 2015. *Oxford Bilingual School Dictionary: isiZulu and English / Isichazamazwi Sesikole Esinezilimi Ezimbili: IsiZulu NesiNgisi, Esishicilelwe abakwa-Oxford. Second Edition*. Oxford University Press Southern Africa.

Roald Eiselen and Martin J. Puttkammer. 2014. Developing text resources for ten south african languages. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014, Reykjavik, Iceland, May 26-31, 2014*, pages 3698–3703. European Language Resources Association (ELRA).

Nikhil Gilbert and C. Maria Keet. 2018. Automating question generation and marking of language learning exercises for isizulu. In *Controlled Natural Language - Proceedings of the Sixth International Workshop, CNL 2018, Maynooth, Co. Kildare, Ireland, August 27-28, 2018*, volume 304 of *Frontiers in Artificial Intelligence and Applications*, pages 31–40. IOS Press.

D. Gkatzia, O. Lemon, and V. Rieser. 2016. Natural language generation enhances human decision-making with uncertain information. In *Proceedings of Association for Computational Linguistics 2016, Vol 2: Short Papers*, pages 264–268. Association for Computational Linguistics.

L. Grout. 1893. *The IsiZulu: A Revised Edition of a Grammar of the Zulu Language*. K. Paul, Trench, Trübner.

Robert K. Herbert and Richard Bailey. 2002. *The Bantu languages: sociohistorical perspectives*, page 50–78. Cambridge University Press.

F. Katamba. 2014. Bantu Nominal Morphology. In Derek Nurse and Gérard Philippson, editors, *The Bantu Languages*, chapter 7, pages 103–120. Routledge.

C. M. Keet and L. Khumalo. 2017a. Grammar rules for the isizulu complex verb. *Southern African Journal of Language and Linguistics*, 35(2):183–200.

C. M. Keet and L. Khumalo. 2017b. Toward a knowledge-to-text controlled natural language of isiZulu. *Language Resources and Evaluation*, 51(1):131–157.

C. M. Keet, M. Xakaza, and L. Khumalo. 2017. Verbalising OWL ontologies in isiZulu with Python. In *The Semantic Web: Extended Semantic Web Conference 2017 Satellite Events - Extended Semantic Web Conference 2017 Satellite Events, Portorož, Slovenia, May 28 - June 1, 2017, Revised Selected Papers*, volume 10577 of *Lecture Notes in Computer Science*, pages 59–64. Springer.

Z. Mahlaza. 2018. Grammars for generating isiXhosa and isiZulu weather bulletin verbs. Msc. thesis, Department of Computer Science, University of Cape Town, South Africa.

Z. Mahlaza and C. M. Keet. 2020. OWLSIZ: An isiZulu CNL for structured knowledge validation. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 15–25, Dublin, Ireland (Virtual). ACL.

L. Marais. 2021a. Approximating a Zulu GF concrete syntax with a neural network for natural language understanding. In *Proceedings of the Seventh International Controlled Natural Language Workshop*, pages 29–38, Amsterdam, Netherlands. Association for Computational Linguistics.

Laurette Marais. 2021b. Mburisano Covid-19 multilingual corpus. Data retrieved from South African Centre for Digital Language Resources, `https://hdl.handle.net/20.500.12185/536`.

Laurette Marais, Johannes A. Louw, Jaco Badenhorst, Karen Calteaux, Ilana Wilken, Nina van Niekerk, and Glenn Stein. 2020. AwezaMed: A multilingual, multimodal speech-to-speech translation application for maternal health care. In *IEEE 23rd International Conference on Information Fusion, FUSION 2020, Rustenburg, South Africa, July 6-9, 2020*, pages 1–8. IEEE.

Alireza Mohammadshahi, Vassilina Nikoulina, Alexandre Berard, Caroline Brun, James Henderson, and Laurent Besacier. 2022. Small-100: Introducing shallow multilingual machine translation model for low-resource languages. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 8348–8359. Association for Computational Linguistics.

Junior Moraba. 2021. Development of a finance based IsiZulu NLG system that verbalises numbers in contex. BSc(hons) project report, Department of Computer Science, University of Cape Town, South Africa.

S. Naidoo. 2005. *Intrusive stop formation in Zulu: an application of feature geometry theory*. Ph.D. thesis, Department of African Languages, University of Stellenbosch.

Nobuhle Ndimande-Hlongwa. 2010. Corpus planning, with specific reference to the use of standard isiZulu in media. *Alternation*, 17(1):207–224.

Evander Nyoni and Bruce A. Bassett. 2021. Low-Resource Neural Machine Translation for Southern African Languages. *arXiv e-prints*.

P. E. Raper. 2012. The Zulu language. *Acta Academica*, 2012(sup-2):22–31.

Georg I. Schlünz, Nkosikhona Dlamini, Alfred Tshoane, and Stan Ramunyisi. 2017. Text normalisation in text-to-speech synthesis for south african languages: Native number expansion. In *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*, pages 230–235.

Tshephisho J. Sefara, Skhumbuzo G. Zwane, Nelisiwe Gama, Hlawulani Sibisi, Phillemon N. Senoamadi, and Vukosi Marivate. 2021. Transformer-based machine translation for low-resourced languages embedded with language identification. In *2021 Conference on Information Communications Technology and Society (ICTAS)*, pages 127–132.

S. Smith. 2020. Generating natural language isiZulu text from mathematical expressions. Bachelor's thesis, University of Cape Town.

P. A. Stuart. 1940. *A Zulu grammar for beginners*. Shuter & Shooter.

C. van der Lee, B. Verduijn, E. Krahmer, and S. Wubben. 2018. Evaluating the text quality, human likeness and tailoring component of PASS: A Dutch data-to-text system for soccer. In *Proceedings of the 27th International Conference on Computational Linguistics, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 962–972. Association for Computational Linguistics.

A. Wilkes and N. Nkosi. 2012. *Complete Zulu Beginner to Intermediate Book and Audio Course: Learn to read, write, speak and understand a new language with Teach Yourself.* Hachette UK.

A Linguistic patterns

In this appendix, we list the identified patterns. In the patterns below, the italics denote fixed string segments, bold text denote special elements/slots (variables) and subscripts a constraint on them. Specifically, they indicate the position where the concords and the stems that quantify the number of 10s/100s/1000s must be inserted. The **adj. conc.** denotes that adjectival concord must be inserted, **poss.conc.** the possessive concord, and **stem** for each stem slot. Subscripts are used to distinguish the numerical stems that can be used, which are either for the number of 10s (i.e., **stem**$_{count10}$), 100s (i.e., **stem**$_{count100}$), 100s (i.e., **stem**$_{count1000}$) or the remainder after removing multiples of 10, 100, and 1000 from the number to be verbalised (i.e., **stem**$_{number<10}$). The **bsc. pref.**$_{nm}$ denotes the so-called basic prefix. It is formed by removing the augment and nasals from a noun class's prefix. For instance, you can form the basic prefix from noun class 10's prefix *izin-* by removing the augment *i-* and nasal *-n-* to obtain *-zi-*. Blue text highlights the possible prefixes that can precede the stems inserted into the slots. Bold orange and pink text highlight the multiple fixed segments that can precede the *-shumi* 'ten', *-khulu* 'hundred', and *-nkulungwane* 'thousand' stems.

Regular expression operators have their usual meaning ("?": zero or one; "|": or; brackets for scope). We use dashes to indicate the separation between morphemes[8]. The dashes are not included in the final text and the combination of morphemes to the left and right of dashes may activate phonological conditioning rules.

1. Cardinal numbers:
   (a) *isi*-**stem**$_{number<10}$
   (b) **adj. conc.**-(*ayisi*)?-**stem**$_{number<10}$
   (c) **adj.conc**-(***yi***|***ngama***)-*shumi* ((***ama***|***ayisi***)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?
   (d) **adj.conc**-(***yi***|***ngama***)-*khulu* ((***ama***|***ayisi***)-**stem**$_{count100}$)? (*na*-(***yi***|***ngama***)-*shumi* ((***ama***|***ayisi***)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?
   (e) **adj.conc**-(***yi***|***yizi***)-*nkulungwane* ((***ezin***|***ezim***|***eziyi***)-**stem**$_{count1000}$)? (*na*-(***yi***|***ngama***)-*khulu* ((***ama***|***ayisi***)-**stem**$_{count100}$)? (*na*-(***yi***|***ngama***)-*shumi*

[8]We also use the term 'morpheme' in reference to combined morphemes (e.g., -ngama-), unless the result is a complete word.

((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?)?

(f) (*i*|*ama*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?

(g) (*i*|*ama*)-*khulu* ((*ama*|*ayisi*)-**stem**$_{count100}$)? (*na*-(*i*|*ama*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?

(h) (*i*|*izi*)-*nkulungwane* ((*ezin*|*ezim*|*eziyi*)-**stem**$_{count1000}$)? (*na*-(*i*|*ama*)-*khulu* ((*ama*|*ayisi*)-**stem**$_{count100}$)? (*na*-(*i*|*ama*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?)?

2. Ordinal numbers:
   (a) *isi*-**stem**$_{number<10}$
   (b) **poss. conc.-stem**$_{number<10}$
   (c) **poss.conc**-(*i*|*ma*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?
   (d) **poss.conc**-(*i*|*ma*)-*khulu* ((*ama*|*ayisi*)-**stem**$_{count100}$)? (*na*-(*i*|*ma*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?
   (e) **poss.conc**-(*i*|*izi*)-*nkulungwane* ((*ezin*|*ezim*|*eziyi*)-**stem**$_{count1000}$)? (*na*-(*i*|*ma*)-*khulu* ((*ama*|*ayisi*)-**stem**$_{count100}$)? (*na*-(*i*|*ma*)-*shumi* ((**ama**|**ayisi**)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?)?
   (f) (*i*|*ama*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?
   (g) (*i*|*ama*)-*khulu* ((*ama*|*ayisi*)-**stem**$_{count100}$)? (*na*-(*i*|*ama*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?
   (h) (*i*|*izi*)-*nkulungwane* ((*ezin*|*ezim*|*eziyi*)-**stem**$_{count1000}$)? (*na*-(*i*|*ama*)-*khulu* ((*ama*|*ayisi*)-**stem**$_{count100}$)? (*na*-(*i*|*ama*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?)?
3. Set-of-items numbers:
   (a) **poss. conc.**-*o*-**bsc. pref.**$_{nm}$-(*yisi*)?-**stem**$_{number<10}$
   (b) *isi*-**stem**$_{number<10}$
   (c) **adj.conc**-(*li*|*ma*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?
   (d) **adj.conc**-(*li*|*ma*)-*khulu* ((*ama*|*ayisi*)-**stem**$_{count100}$)? (*na*-(*li*|*ma*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?
   (e) **adj.conc**-y-(*i*|*yizi*)-*nkulungwane* ((*ezin*|*ezim*|*eziyi*)-**stem**$_{count1000}$)? (*na*-(*li*|*ma*)-*khulu* ((*ama*|*ayisi*)-**stem**$_{count100}$)? (*na*-(*li*|*ma*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?)?
   (f) (*i*|*ama*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?
   (g) (*i*|*ama*)-*khulu* ((*ama*|*ayisi*)-**stem**$_{count100}$)? (*na*-(*i*|*ama*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?
   (h) (*i*|*izi*)-*nkulungwane* ((*ezin*|*ezim*|*eziyi*)-**stem**$_{count1000}$)? (*na*-(*i*|*ama*)-*khulu* ((*ama*|*ayisi*)-**stem**$_{count100}$)? (*na*-(*i*|*ama*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?)?
4. Adverbs:
   (a) *ka*-(si)?-**stem**$_{number<10}$
   (b) *kali*-(*shumi*|*khulu*)
   (c) *kayi-nkulungwane*
   (d) (*kali*|*kanga*)-(*i*|*ama*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?
   (e) (*kali*|*kanga*)-(*i*|*ama*)-*khulu* ((*ama*|*ayisi*)-**stem**$_{count100}$)? (*na*-(*i*|*ama*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**)))?
   (f) *kayi*-(*i*|*izi*)-*nkulungwane* ((*ezin*|*ezim*|*eziyi*)-**stem**$_{count1000}$)? (*na*-(*i*|*ama*)-*khulu* ((*ama*|*ayisi*)-**stem**$_{count100}$)? (*na*-(*i*|*ama*)-*shumi* ((*ama*|*ayisi*)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?)?

B Pattern use rules

The conditions for when to select each pattern are based on the range of the number:

1. **Range:** $0 < n < 10$, **Patterns:** 1a, 1b, 2a, 2b, 3a, 3b, 4a, **Comment:** The patterns 1a, 2a, and 3b are used when the number must not include an agreement marker and 1b, 2b, and 3a are used when such a marker must exist. 4a is used for numbers below ten and its optional segment is only included for values in the inclusive range [6-9].
2. **Range:** $10 \leq n < 100$, **Patterns:** 1c, 1f, 2c, 2f, 3c, 3f, 4b, 4d, **Comment:** The patterns 1f, 2f, 3f are used when there are no agreement

markers while 1c, 2c, 3c are used when such markers exist. The adverb pattern 4b is used when the number is 10 and 4d is used when the number is greater than 10.

3. **Range:** $100 \leq n < 1000$, **Patterns:** 1d, 1g, 2d, 2g, 3d, 3g, 4b, 4e, **Comment:** The patterns 1g, 2g, and 3g is used when there are no agreement markers while 1d, 2d, and 3d is used when such markers exist. The adverb pattern 4b is used when the number is 100 and 4e is used when the number is greater than 100. 4e is used when there are multiple 100s while 3e is used when there is a single 100.
4. **Range:** $1000 \leq n < 10000$, **Patterns:** 1e, 1h, 2e, 2h, 3e, 3h, 4c, 4f, **Comment:** The patterns 1h, 2h, and 3h is used when there are no agreement markers while 1e, 2e, and 3e is used when such markers exist. The adverb pattern 4c is used when the number is 1000 and 4f it used when the number is greater than 1000. 4f is used when there are multiple 1000s while 4c is used when there is a single 1000.

We now turn to list the rules used to select one of many optional segments that can be found in each pattern:

1. The stems *-shumi* 'ten', *-khulu* 'hundred', and *-nkulungwane* 'thousand' are preceded by **Segment 1** and/or **Segment 2** and values must be chosen for both segments. For instance, when forming the first word in Pattern 4f, we must choose either **-i-** or **-izi-** and append it to the leading *kayi-*. Linguistically, the leading prefix *kayi-* is formed by combining the adverbial prefix *ka-* and copula *-yi-*. The rules for selecting the appropriate prefix value for every multiple of ten that has a unique stem (i.e., 10, 100, and 1000) are listed in Table 2. The value depends on the category of the number and whether there is a single or multiple of tens. To demonstrate how to use the rules in Table 2, consider the verbalisation of the adverbial number 3333 in a sentence where its subject is *izincwadi* 'books' — a noun from class 8: *Ngithenge izincwadi* ***kayizinkulungwane*** *ezinthathu* ***namakhulu*** *amathathu* ***namashumi*** *amathathu nantathu.* 'I bought books three thousand three hundred and thirty-three times'.
   The adverb category does not have patterns with agreement markers even though the numbers may be used in situations where they have a noun as a subject. As can be seen in the patterns, all such numbers begin with the adverbial prefix *ka-* and it is either followed by the basic prefix for noun class 7 *-si-*, basic prefix for noun class 5 *-li-*, the copula *-yi-*, or the adverb prefix *-nga-*. The choice of which morpheme to append to *ka-* depends on whether the input is less than 10 (uses *-si-*), equal to 10/100 (uses *-li-*) or 1000 (uses *-yi-*), or is a multiple of 10/100/1000 that has a remainder after removing the 10/100/1000s (10 and 100s uses *-li-* and *-nga-* while 1000 use *-yi-*). In the 3333 case, since there are multiple 1000s in the number 3333 then pattern 4f is chosen. The first two words in the isiZulu sentence (i.e., *Ngithenge izincwadi*) mean 'I bought books' so our explanation will not focus on them. For the number 3333, from left to right, there is first the 3000-part and its first morpheme for every input has the value *kayi-* and it is formed by combining the *ka-* adverbial prefix with the copulative *-yi-*. To get the value of the second morpheme of the word we use Table 2 to select an appropriate morpheme: the 1000 column, plural, adverb, so the **-izi-** prefix is chosen. Then the prefix for the word is formed by combining *kayi-+-izi-* to obtain *kayizi-* instead of *kayiizi-* since the second *-i-* is eliminated by phonological conditioning rules. The first word is then formed by combining the prefix *kayizi-* and stem *-nkulungwane* to form the word *kayizinkulungwane*. The formation of second word in the pattern, the three of those thousands to result in *ezinthathu*, is explained in the next item.
2. The words that quantify the exact number of 10s, 100s, and 1000s are also preceded by prefixes. To demonstrate, consider the formation of the underlined word in *amakhulu amabili* 'two hundred' (formed using Pattern 1h). Generally, the prefix for these words is chosen according to the rules specified in Table 3. Linguistically, the difference between the ranges 2-5 and 6-9, shown in the table, is that the 2-5 range forms words by combining noun class 6's adjectival concord—the noun forms of the input belong it—*ama-* with the stem (i.e., *-bili* 'two', *-thathu* 'three', *-ne* 'four', and *-hlanu* 'five' respectively) while the 6-9 range combines noun class 6's augment *a-*, the copula *-yi-*, and noun form of input (i.e., *isithupa* 'six', *isikhombisa* 'seven', *isishiyagalombili* 'eight', and *isishiyagalolunye* 'nine'

respectively). For the 1000s in Table 3, the difference is partially determined by phonological conditioning. Returning to the verbalisation output of *amakhulu* <u>*amabili*</u> ‘two hundred’, for 2 of hundreds, **ama-** is selected (3rd row). For the 3 of thousands of the previous 3333 example, it is **ezin-** (5th row), which is then added to the number 3, *-thathu* to make *ezinthathu*.

3. The segments that quantify the number of 10/100/1000s can be part of larger optional segments (e.g., Pattern 4f's first optional segment ((***ezin***|***ezim***|***eziyi***)-**stem**$_{count1000}$)?). These are only included if the input number has multiple values of 10/100/1000 after removing larger multiples of 10. The last optional segment is only included if there a remainder after removing all the multiples of 10, 100, and 1000 from the input. For instance, when verbalising the cardinal number 321 using Pattern 1g, the last optional segment ((***ama***|***ayisi***)-**stem**$_{count10}$)? is included since there are two multiples of 10 in the number (i.e., 20) after removing the three multiples of 100 (i.e., 300). Similarly, the last optional segment (*na*-(**stem**$_{number<10}$ | **noun**))? is included since there is a remainder of 1 after removing all the multiples of 10.

Table 2 lists the rules used to select the possible prefix values that are used in constructing the strings that refer to special multiples of ten. Table 3 lists rules for constructing the prefixes used when forming the words for counting the number of multiples of 10s.

C Pattern updates

The updates made to the patterns and rules for their use after the evaluation are included in Table 7.

D Final algorithms

The algorithms rely on several helper functions: $getStem$, $getPrefix$, $getWord$, $getWordCount$, and $getNoun$. The $getStem$ function is responsible for retrieving the stem for all supported numbers. The stems for all such numbers are as follows: *-nye* (1), *-bili* (2), *-thathu* (3), *-ne* (4), *-hlanu* (5), *-thupha* (6), *-khombisa* (7), *-shiyagalombili* (8), *-shiyagalolunye* (9), *-shumi* (10), *-khulu* (100), and *-nkulungwane* (1000). The $getPrefix$ and $getWord$ functions work together to encode the rules specified in Table 2 and concatenating the second segment to the appropriate stem, the $getWordCount$ function constructs the word for counting the multiples of 10/100/1000, and the $getNoun$ function is responsible for constructing a noun from a number's stem by either prefixing *i-* in the case of 10 or *isi-* otherwise.

We illustrate the algorithms by demonstrating the verbalization of the cardinal number 22 (with and without agreement markers). We begin by demonstrating the verbalisation of the number when there are no agreement markers. Algorithm 1 starts by initialising the string (line 2), it then resolves that nearest multiple of 10 is just 10 (lines 20-22) with a remainder of 2 (line 23) after subtracting all the appropriate multiples 10. It then determines that there are two multiples of 10 in the input (line 24) and then constructs the initial value of the verbalised string to take the form *amashumi* (line 29). Since there are multiple 10s in 20 (line 31), it uses the $getWordCount$ method to construct the word *amabili* ‘two’ and that is appended to current form of the final string (line 32). The existing remainder (lines 34 and 35) is less than six, so it resolves its stem *-bili* ‘two’ and appends it to the conjunction *na-* (line 36). The combination of *na-* and *-bili* activates phonological conditioning rules which introduce an *-m-* between the two segments. This entire process then produces the verbalised string *amashumi amabili nambili* ‘two tens and two’, i.e., ‘twenty-two’.

To demonstrate the verbalisation of the cardinal number 22 when it agrees with any noun in class 8, the final algorithm for such cases (i.e., Algorithm 2) starts by initialising an empty string (line 2) and like Algorithm 1, it then resolves that nearest multiple of 10 is just 10 (lines 16-18) with a remainder of 2 after subtracting all the appropriate multiples of 10 (line 19). Since the category of the input number is cardinal, it retrieves the adjectival concord *ezi-* to use as a prefix (line 22-23) and appends it together with the segment *-ngama-*, retrieved using $getPrefix$ using the rules defined in Table 2, to the stem to form *ezingamashumi* (line 29). Since there are two 10s in 20, it then uses $getWordCount$ to form *amabili* ‘two’ (line 21). After that, it then fetches the stem for the remainder (line 35) and appends it to the conjunction *na-* to form *nambili* since the remainder of 2 is less than 6. Finally, the algorithm then produces the text *ezingamashumi amabili nambili* ‘twenty-two’. The difference between the evaluated algorithm (i.e.,

Algorithm 1 Numbers with no agreement markers

```
1: verbalise (number, category):
2: $s \leftarrow \varnothing$    ▷ Initialising the verbalised string
3: $uss = [s_1, s_2, ..., s_n]$    ▷ Numbers with unique stems (e.g., 1 = -nye, 2 = -bili)
4: if $category = cardinal$ and $number < 10$ then    ▷ Verbalise cardinals that are less than 10
5:     $s \leftarrow$ isi $+ getStem(number)$    ▷ Attach isi- to stem
6: else if $category = adverb$ and $number \in uss$ then    ▷ Verbalise adverbs with unique stems
7:     if $0 < number < 6$ then
8:         $s \leftarrow$ka$+getStem(number)$    ▷ Attach ka- to stem
9:     else if $5 < number < 10$ then
10:        $s \leftarrow$kasi$+getStem(number)$    ▷ Attach kasi- to stem
11:    else if $number = 10$ or $100$ then
12:        $s \leftarrow$kali$+getStem(number)$    ▷ Attach kali- to stem
13:    else if $number = 1000$ then
14:        $s \leftarrow$kayi$+getStem(number)$    ▷ Attach kayi- to stem
15:    end if
16: else
17:    $uts = [u_1, u_2, ..., u_m]$    ▷ Multiples of 10 with unique stems (e.g., 10 = -shumi, 100 = khulu)
18:    for $u_i \in uts$ do
19:        if $u_i > number$ then    ▷ First multiple of 10 with unique stem > current number
20:            $nearest = u_{i-1}$    ▷ Last multiple of 10 with unique stem < current number
21:            $remainder = number$ mod $nearest$    ▷ Remainder after removing multiples of nearest 10s
22:            $nv = (number - remainder)/nearest$    ▷ Count of multiples of the nearest 10s
23:            $p = nv > 1$    ▷ Determining whether there are ≥1 multiples of 10s
24:            if $category = adverb$ then    ▷ Verbalising first word for the adverb
25:                $s \leftarrow getPrefix(nearest, category, p) + getWord(nearest, p)$    ▷ Segments 1, 2 + stem
26:            else
27:                $s \leftarrow getWord(nearest, p)$    ▷ Attach Segment 2 + stem
28:            end if
29:            if $p$ then    ▷ Verbalise second word if there are multiple 10s
30:                $s \leftarrow s +' ' + getWordCount(nv, nearest)$    ▷ Attach Table 3 prefix + stem
31:            end if
32:            if $remainder > 0$ then    ▷ Verbalise last segment if there is a remainder
33:                if $remainder < 6$ then
34:                    $s \leftarrow s +' ' + na + getStem(remainder)$    ▷ Attach na+stem for numbers <six
35:                else if $5 < remainder < 10$ then
36:                    $s \leftarrow s +' ' + na + getNoun(remainder)$    ▷ Attach na+noun for other numbers <10
37:                else
38:                    $s \leftarrow s +' ' + na + verbalise(remainder, category)$    ▷ Recursively, verbalise ≥10
39:                end if
40:            end if
41:        end if
42:    end for
43: end if
44: Return $s$
```

Algorithm 2 Numbers with agreement markers

```
verbalise (number, category, nc):
$s \leftarrow \varnothing$    ▷ Initialising the verbalised string
$uss = [s_1, s_2, ..., s_n]$    ▷ Numbers with unique stems (e.g., 1 = -*nye*, 2 = -*bili*)
**if** $category = cardinal$ and $number < 10$ **then**
    $s \leftarrow getAdjC(nc) + getStem(number)$    ▷ Attach adjectival concord for cardinals < 10
**else if** $category = ordinal$ and $number \in uss$ **then**
    $s \leftarrow getPossC(nc) + getStem(number)$    ▷ Attach poss. concord for ordinals with unique stems
**else if** $category = set\text{-}of\text{-}items$ and $number < 10$ **then**    ▷ Using Pattern 3a
    **if** $5 < number < 10$ **then**
        $s \leftarrow getPossC(nc)$+o+$getBasPref_{nm}(nc)$+yisi+$getStem(number)$    ▷ Include -*yisi*-
    **else**
        $s \leftarrow getPossC(nc)$+o+$getBasPref_{nm}(nc) + getStem(number)$    ▷ Do not include -*yisi*-
    **end if**
**else**
    $uts = [u_1, u_2, ..., u_m]$    ▷ Multiples of 10 with unique stems (e.g., 10 = -*shumi*, 100 = *khulu*)
    **for** $u_i \in uts$ **do**
        **if** $u_i > number$ **then**    ▷ First multiple of 10 with unique stem > current number
            $nearest = u_{i-1}$    ▷ Last multiple of 10 with unique stem < current number
            $remainder = number$ **mod** $nearest$    ▷ Remainder after removing multiples of nearest 10s
            $nv = (number - remainder)/nearest$    ▷ Count of multiples of the nearest 10s
            $p = nv > 1$    ▷ Determining whether there are ≥1 multiples of 10s
            **if** $category = cardinal$ **then**
                $s \leftarrow getAdjC(nc)$    ▷ Attach adjectival concord
            **else if** $category = ordinal$ **then**
                $s \leftarrow getPossC(nc)$    ▷ Attach possessive concord
            **else if** $category = set\text{-}of\text{-}items$ **then**
                $s \leftarrow getPossC(nc)$+o+$getBasPref_{nm}(nc)$    ▷ Attach poss. concord and basic prefix
            **end if**
            $s \leftarrow s + getPrefix(nearest, category, p) + getStem(nearest, p)$    ▷ Attach Segm. 2 + stem
            **if** $p$ **then**    ▷ Verbalise second word if there are multiple 10s
                $s \leftarrow s +' \ ' + getWordCount(nv, nearest)$    ▷ Attach Table 3 prefix + stem
            **end if**
            **if** $remainder > 0$ **then**
                **if** $remainder < 6$ **then**
                    $s \leftarrow s +' \ ' + na + getStem(remainder)$    ▷ Attach na+stem for numbers <six
                **else if** $5 < remainder < 10$ **then**
                    $s \leftarrow s +' \ ' + na + getNoun(remainder)$    ▷ Attach na+noun for other numbers <10
                **else**
                    $s \leftarrow s +' \ ' + na + verbalise(remainder, category)$    ▷ Recursively, verbalise nums. ≥10
                **end if**
            **end if**
        **end if**
    **end for**
**end if**
**Return** $s$
```

Table 7: List of updated patterns for set-of-items numbers.

Pattern identifier	**Pattern**
Evaluated 3c	**adj.conc**-(***li***|***ma***)-*shumi* ((***ama***|***ayisi***)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?
Corrected 3c	**poss.conc.**-*o*-**bsc.pref.**$_{nm}$-(***yi***|***ngama***)-*shumi* ((***ama***|***ayisi***)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?
Evaluated 3d	**adj.conc**-(***li***|***ma***)-*khulu* ((***ama***|***ayisi***)-**stem**$_{count100}$)? (*na*-(***li***|***ma***)-*shumi* ((***ama***|***ayisi***)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?
Corrected 3d	**poss.conc.**-*o*-**bsc.pref.**$_{nm}$-(***yi***|***ngama***)-*khulu* ((***ama***|***ayisi***)-**stem**$_{count100}$)? (*na*-(***li***|***ma***)-*shumi* ((***ama***|***ayisi***)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?
Evaluated 3e	**adj.conc**-y-(***i***|***yizi***)-*nkulungwane* ((***ezin***|***ezim***|***eziyi***)-**stem**$_{count1000}$)? (*na*-(***li***|***ma***)-*khulu* ((***ama***|***ayisi***)-**stem**$_{count100}$)? (*na*-(***li***|***ma***)-*shumi* ((***ama***|***ayisi***)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?)?
Corrected 3e	**poss.conc.**-*o*-**bsc.pref.**$_{nm}$-(***yi***|***yizi***)–*nkulungwane* ((***ezin***|***ezim***|***eziyi***)-**stem**$_{count1000}$)? (*na*-(***li***|***ma***)-*khulu* ((***ama***|***ayisi***)-**stem**$_{count100}$)? (*na*-(***li***|***ma***)-*shumi* ((***ama***|***ayisi***)-**stem**$_{count10}$)? (*na*-(**stem**$_{number<10}$ | **noun**))?)?)?

Algo 3 in Appendix E) and final algorithm pertains to set-of-items numbers and will be discussed in Section 5.

E Evaluated algorithm

The evaluated algorithm for verbalising numbers when they agree with a noun is listed in Algorithm 3.

Algorithm 3 Evaluated algorithm for verbalising numbers with agreement markers

```
verbalise (number, category, nc):
$s \leftarrow \varnothing$    ▷ Initialising the verbalised string
$uss = [s_1, s_2, ..., s_n]$    ▷ Numbers with unique stems (e.g., 1 = -nye, 2 = -bili)
if $category = cardinal$ and $number < 10$ then
    $s \leftarrow getAdjC(nc) + getStem(number)$    ▷ Attach adjectival concord for cardinals < 10
else if $category = ordinal$ and $number \in uss$ then
    $s \leftarrow getPossC(nc) + getStem(number)$    ▷ Attach poss. concord for ordinals with unique stems
else if $category = set\text{-}of\text{-}items$ and $number < 10$ then    ▷ Using Pattern 3a
    if $5 < number < 10$ then
        $s \leftarrow getPossC(nc)$+o+$getBasPref_{nm}(nc)$+yisi+$getStem(number)$    ▷ Include -yisi-
    else
        $s \leftarrow getPossC(nc)$+o+$getBasPref_{nm}(nc) + getStem(number)$    ▷ Do not include -yisi-
    end if
else
    $uts = [u_1, u_2, ..., u_m]$    ▷ Multiples of 10 with unique stems (e.g., 10 = -shumi, 100 = khulu)
    for $u_i \in uts$ do
        if $u_i > number$ then    ▷ First multiple of 10 with unique stem > current number
            $nearest = u_{i-1}$    ▷ Last multiple of 10 with unique stem < current number
            $remainder = number$ mod $nearest$    ▷ Remainder after removing multiples of nearest 10s
            $nv = (number - remainder)/nearest$    ▷ Count of multiples of the nearest 10s
            $p = nv > 1$    ▷ Determining whether there are ≥1 multiples of 10s
            if $category = cardinal$ or $set\text{-}of\text{-}items$ then
                $s \leftarrow getAdjC(nc)$    ▷ Attach adjectival concord
            else if $category = ordinal$ then
                $s \leftarrow getPossC(nc)$    ▷ Attach possessive concord
            end if
            $s \leftarrow s + getPrefix(nearest, category, p) + getStem(nearest, p)$    ▷ Attach Segm. 2 + stem
            if $p$ then    ▷ Verbalise second word if there are multiple 10s
                $s \leftarrow s +' ' + getWordCount(nv, nearest)$    ▷ Attach Table 3 prefix + stem
            end if
            if $remainder > 0$ then
                if $remainder < 6$ then
                    $s \leftarrow s +' ' + na + getStem(remainder)$    ▷ Attach na+stem for numbers <six
                else if $5 < remainder < 10$ then
                    $s \leftarrow s +' ' + na + getNoun(remainder)$    ▷ Attach na+noun for other numbers <10
                else
                    $s \leftarrow s +' ' + na + verbalise(remainder, category)$    ▷ Recursively, verbalise nums. ≥10
                end if
            end if
        end if
    end for
end if
Return $s$
```